Computer Science 409, Spring 2001: Prelim Review

1. Suppose $T(n) = T(\lfloor an \rfloor) + T(\lfloor bn \rfloor) + \lfloor cn \rfloor$, where $a + b < 1$. Show that $T(n) = O(n)$. (Hint: use induction.)

2. You have a hash table of size $m = 11$ and hash functions h_1 and h_2

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod (m - 1)).$$

   (a) Show the contents of the hash table after inserting, in order, the keys 10, 22, 32, 4, 15, using chaining and h_1 only.

   (b) Show the contents of the hash table after inserting, in order, the keys 10, 22, 32, 4, 15, using **open addressing with double hashing**, and $h_1(k)$ and $h_2(k)$ above, as hash functions.

3. Assume you know that there will be no more than n different keys, and want to use open addressing with double hashing. What table size and what functions you would use?

4. (a) State the *heap property*. (Assume we want the biggest element at the root of the heap.)

   (b) Construct a heap containing the following keys: 1,2,9,11,12, 15.

   (c) What happens to the in part (b) after an EXTRACT-MAX.

   (d) Give $O(\log n)$ algorithms (either in understandable pseudocode or explain how it works) to *increase* the key of an element, and to *decrease* the key of an element in a heap. (Here n denotes the number of elements in the Heap.) (We will need DECREASE-KEY in Dijkstra's algorithm and Prim's algorithm, which will be covered shortly.)

5. Suppose you have an application of priority queues where you know the that there can be only k different priorities, called priorities $1, \ldots, k$. That is, an item is a pair (p, x), where p is the priority (which is in $\{1, \ldots, k\}$) and x is a pointer to a record. You know that there will be a large number N of items in your priority queue, where N is much bigger than k, so many items will have the same priority.

   (a) Suppose you use a standard heap-based priority queue and put the N elements in this priority queue. What is the worst case running time of EXTRACT-MAX and INSERT in this implementation? (In this case, you have to insert the pair (p, x), but the heap property applies only to the p component. For EXTRACT-MAX, you can extract any element of maximum priority.)

(b) Design a data structure for this application that allows the INSERT and EXTRACT-MAX operations to run in $O(\log k)$ time. You may use any data structure that we learned about without writing code for it, but be careful in explaining any modification you must make for this problem. Explain clearly how you do `insert` and EXTRACT-MAX. You may use any data structure operations that we learned about without writing code for them.

6. **Implementing a data base operation:**

A database often consists of data which is in the form of records. For example, the ACM (Association of Computing Machinery) membership database might include lists of records of the members of the different subgroups of ACM. (These subgroups are called SIGs, or Special Interest Group.) Here's an example of a very small database of this form:

SIGGRAPH	**SIGCOMM**
Jones	Wilson
Chang	Jones
Markowitz	Smith
Brown	Chang

A standard database operation is the *join* operation: Given two lists of records, find their common members. So, for example, given the database above, the join of the SIGCOMM members and the SIGGRAPH members is the list (Jones, Chang).

In this problem, we compare two methods of implementing this operation. Assume for simplicity that we consider lists of numbers (rather than items with keys) and the lists have no duplicates. One list is stored in an array `COMM` of length N, and the other list in an array `GRAPH` of length M, and we want to have the numbers that occur in both listed in an array `SIG`. Assume $N > M$. **Give the expected running times of each of the methods using the $O(.)$ notation and the parameters N and M.** Explain your answer.

(a) Select a good hash function, h, and a table size, L, close to M. Use this hash function h to hash the elements of the shorter list, `GRAPH`, into a new table, `H-GRAPH`, using chaining to resolve collisions. Now, go though the other list, `COMM` (with a single `for` loop). Use the search method of hashing to look for each of the numbers in the `COMM` list to check if they also occur on the `H-GRAPH` hash table; if they do, copy them into the `SIG` array.

(b) Build a binary search tree from the items in the shorter list `GRAPH`. Now, go though the other list, `COMM` (with a single `for` loop). Use the search method of BST to look for each of the numbers in the `COMM` list to check if they also occur in the BST; if they do, copy them into the `SIG` array. Is it better in this method to build the BST from the shorter list, or should we build the BST using the longer list and then search for each element of the longer list in this BST? Explain your answer.

7. What is $\log^*(17)$?

8. Order the following running times from fastest to slowest. All logarithms are base 2. No proof or other explanation is necessary. $O(n)$, $O(1)$, $O(n^2)$, $O(\log n)$, $O(n \log n)$, $O(n/(\log n))$.

9. What is a dense graph?